Relatório do EP3
MAC0352 – Redes de Computadores e Sistemas Distribuídos – 2/2019

Luiz Girotto
N°USP: 8941189

Victor Chiaradia Gramuglia Araujo
N°USP: 9793756

11/11/2019

1 Passo 0

Na definição do protocolo OpenFlow, o que um switch faz toda vez que ele recebe um pacote que ele nunca recebeu antes?

Ao receber um pacote que ele nunca recebeu antes, um switch *OpenFlow* encaminha este pacote para o controlador *OpenFlow*. Este então determina a forma com que será lidado este pacote.

2 Passo 2

Com o acesso à Internet funcionando em sua rede local, instale na VM o programa `traceroute` usando `sudo apt install traceroute` e escreva abaixo a saída do comando `sudo traceroute -I www.inria.fr`. Pela saída do comando, a partir de qual salto os pacotes alcançaram um roteador na Europa? Como você chegou a essa conclusão?

```
<traceroute to www.inria.fr (128.93.162.84), 30 hops max, 60 byte packets
 1  10.0.2.2 (10.0.2.2)  0.110 ms  0.095 ms  0.167 ms
 2  192.168.15.1 (192.168.15.1)  1.193 ms  1.457 ms  1.367 ms
 3  * * *
 4  187-100-163-72.dsl.telesp.net.br (187.100.163.72)  50.045 ms  49.909 ms  49$
 5  152-255-158-36.user.vivozap.com.br (152.255.158.36)  7.161 ms  7.071 ms  6.$
 6  84.16.9.109 (84.16.9.109)  13.040 ms  9.830 ms  9.600 ms
 7  94.142.98.177 (94.142.98.177)  124.261 ms  120.983 ms  125.332 ms
 8  84.16.15.129 (84.16.15.129)  120.564 ms  124.784 ms  124.723 ms
 9  213.140.36.89 (213.140.36.89)  124.474 ms  124.306 ms  124.158 ms
10  ip4.gtt.net (208.116.240.149)  145.831 ms  155.251 ms  155.017 ms
11  et-3-3-0.cr4-par7.ip4.gtt.net (213.200.119.214)  230.782 ms  228.760 ms  22$
12  renater-gw-ix1.gtt.net (77.67.123.206)  244.947 ms  246.479 ms  246.284 ms
13  te1-1-inria-rtr-021.noc.renater.fr (193.51.177.107)  240.280 ms  240.100 ms$
14  inria-rocquencourt-te1-4-inria-rtr-021.noc.renater.fr (193.51.184.177)  239$
15  unit240-reth1-vfw-ext-dc1.inria.fr (192.93.122.19)  242.478 ms  241.464 ms $
16  ezp3.inria.fr (128.93.162.84)  238.068 ms  243.845 ms  244.529 ms>
```

A partir do décimo-primeiro salto, os pacotes alcançaram um roteador na Europa. É possível perceber isto pelo tempo de resposta similar ao pacote encontrado no décimo-terceiro salto, o qual possui identificador de endereços da França (fr).

3 Passo 3 - Parte 1

Execute o comando `iperf`, conforme descrito no tutorial, antes de usar a opção `--switch user`, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado).

```
1:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['16.7 Gbits/sec', '16.8 Gbits/sec']

2:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['11.0 Gbits/sec', '11.1 Gbits/sec']

3:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['12.5 Gbits/sec', '12.5 Gbits/sec']

4:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['16.8 Gbits/sec', '16.8 Gbits/sec']

5:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['17.2 Gbits/sec', '17.2 Gbits/sec']
```

Média: 14.84 Gbits/sec. Intervalo de confiança: 95% CI [12.6, 17.1] Gbits/sec

4 Passo 3 - Parte 2

Execute o comando `iperf`, conforme descrito no tutorial, com a opção `--switch user`, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes menos o da Seção anterior? Qual o motivo dessa diferença?

```
1:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['274 Mbits/sec', '275 Mbits/sec']

2:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['274 Mbits/sec', '275 Mbits/sec']
```

```
3:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['275 Mbits/sec', '277 Mbits/sec']

4:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['272 Mbits/sec', '273 Mbits/sec']

5:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['275 Mbits/sec', '277 Mbits/sec']
```

Média: 274 Mbits/sec. Intervalo de confiança: 95% CI [273, 275] Mbits/sec

A largura de banda é aproximadamente 54 vezes menor do que nos testes da seção anterior. Isto ocorre pois agora, ao invés dos pacotes serem trocados apenas por via do *kernel*, elas devem iniciar seu caminho *user-space*, passando então para o *kernel*, e novamente para o *user-space*, o que aumenta bastante o tempo de transmissão dos pacotes.

5 Passo 4 - Parte 1

Execute o comando `iperf`, conforme descrito no tutorial, usando o controlador `of_tutorial.py` original sem modificação, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes menos o da Seção 3? Qual o motivo para essa diferença? Use a saída do comando `tcpdump`, deixando claro em quais computadores virtuais ele foi executado, para justificar a sua resposta.

```
1:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['20.4 Mbits/sec', '22.7 Mbits/sec']

2:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['23.2 Mbits/sec', '26.6 Mbits/sec']

3:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['23.6 Mbits/sec', '27.0 Mbits/sec']

4:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['22.8 Mbits/sec', '26.0 Mbits/sec']

5:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.2 Mbits/sec', '21.7 Mbits/sec']
```

Média: 21.84 Mbits/sec. Intervalo de confiança: 95% CI [20.3, 23.4] Mbits/sec

A largura de banda é aproximadamente 679 vezes menor do que nos testes da seção 3. Isto ocorre pois agora os pacotes são enviados para o controlador, o qual está configurado para comportar-se como um *hub*. Ou seja, todos os pacotes são enviados para todos os computadores conectados a este *hub*, com exceção do computador que enviou o pacote, o que aumenta ainda mais o tempo de transmissão dos pacotes.
O comportamento descrito anteriormente pode ser observado em um experimento feito anteriormente no tutorial, em que um *ping* feito por uma das máquinas era capturado no *output* do `tcpdump` de todas outras máquinas (inclusive as completamente não-relacionadas com o ping). O experimento foi feito utilizando-se de três computadores virtuais, *h1*, *h2* e *h3*. Observe os *outputs* dos computadores *h2* e *h3*, rodando o `tcpdump`, quando o computador virtual *h1* manda um *ping* para o computador *h2*:

h2:

```
13:42:34.525376 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 3129, seq 1, len
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 69f2 4000 4001 bcb4 0a00 0001 0a00  .Ti.@.@.........
        0x0020:  0002 0800 2035 0c39 0001 3aea bd5d 0000  .....5.9..:..]..
        0x0030:  0000 0d76 0700 0000 0000 1011 1213 1415  ...v............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
13:42:34.525423 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 3129, seq 1, lengt
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 f464 0000 4001 7242 0a00 0002 0a00  .T.d..@.rB......
        0x0020:  0001 0000 2835 0c39 0001 3aea bd5d 0000  ....(5.9..:..]..
        0x0030:  0000 0d76 0700 0000 0000 1011 1213 1415  ...v............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
13:42:39.523727 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
13:42:39.523769 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
```

h3:

```
13:42:34.525371 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 3129, seq 1, len
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 69f2 4000 4001 bcb4 0a00 0001 0a00  .Ti.@.@.........
        0x0020:  0002 0800 2035 0c39 0001 3aea bd5d 0000  .....5.9..:..]..
        0x0030:  0000 0d76 0700 0000 0000 1011 1213 1415  ...v............
```

```
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ..........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
13:42:34.527365 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 3129, seq 1, lengt
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 f464 0000 4001 7242 0a00 0002 0a00  .T.d..@.rB......
        0x0020:  0001 0000 2835 0c39 0001 3aea bd5d 0000  ....(5.9..:..]..
        0x0030:  0000 0d76 0700 0000 0000 1011 1213 1415  ...v............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ..........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
13:42:39.523721 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
13:42:39.525462 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
```

Observe que os mesmos pacotes enviados por *h1* foram recebidos por *h2* e *h3*.

6 Passo 4 - Parte 2

Execute o comando `iperf`, conforme descrito no tutorial, usando o seu controlador `switch.py`, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes mais o da Seção anterior? Qual o motivo dessa diferença? Use a saída do comando `tcpdump`, deixando claro em quais computadores virtuais ele foi executado, para justificar a sua resposta.

```
1:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['21.4 Mbits/sec', '24.8 Mbits/sec']

2:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['21.5 Mbits/sec', '24.5 Mbits/sec']

3:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['21.1 Mbits/sec', '23.7 Mbits/sec']

4:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.6 Mbits/sec', '22.7 Mbits/sec']
```

```
5:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['16.9 Mbits/sec', '18.9 Mbits/sec']
```

Média: 20.1 Mbits/sec. Intervalo de confiança: 95% CI [18.6, 21.6] Mbits/sec

A largura de banda é essencialmente a mesma da seção anterior. Este comportamento é esperado pois, por mais que agora os pacotes estejam sendo direcionados somente para os destinos apropriados, os pacotes ainda estão sendo enviados para o controlador para serem processados, o que faz com que a largura de banda não seja melhorada significativamente. De qualquer forma, o novo comportamento determinado pelo controlador pode ser observado nos outputs do `tcpdump` dos computadores *h2* e *h3*:

h2:

```
07:01:33.991436 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
07:01:33.991494 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
07:01:33.998444 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7453, seq 1, len
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 4649 4000 4001 e05d 0a00 0001 0a00  .TFI@.@..]......
        0x0020:  0002 0800 6040 1d1d 0001 cdeb be5d 0000  ....`@.......]..
        0x0030:  0000 2185 0e00 0000 0000 1011 1213 1415  ..!.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
07:01:33.998474 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7453, seq 1, lengt
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 186a 0000 4001 4e3d 0a00 0002 0a00  .T.j..@.N=......
        0x0020:  0001 0000 6840 1d1d 0001 cdeb be5d 0000  ....h@.......]..
        0x0030:  0000 2185 0e00 0000 0000 1011 1213 1415  ..!.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
07:01:39.008088 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0000 0a00 0001                 ..........
07:01:39.064899 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0002 0a00 0002                 ..........
```

```
h3:

07:01:33.991433 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
```

Nota-se que o computador *h2*, alvo do `ping` de *h1*, recebeu todos os pacotes relevantes a este, enquanto *h3* recebeu apenas o pacote `ARP`, de *broadcast*.

7 Passo 4 - Parte 3

Execute o comando `iperf`, conforme descrito no tutorial, usando o seu controlador `switch.py` melhorado, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes mais o da Seção anterior? Qual o motivo dessa diferença? Use a saída do comando `tcpdump`, deixando claro em quais computadores virtuais ele foi executado, e saídas do comando `sudo ovs-ofctl`, com os devidos parâmetros, para justificar a sua resposta.

```
1:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.4 Gbits/sec', '19.5 Gbits/sec']

2:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.7 Gbits/sec', '19.7 Gbits/sec']

3:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.2 Gbits/sec', '19.3 Gbits/sec']

4:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.4 Gbits/sec', '19.4 Gbits/sec'

5:
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['19.6 Gbits/sec', '19.6 Gbits/sec']]
```

Média: 19.46 Gbits/sec. Intervalo de confiança: 95% CI [19.3, 19.6] Gbits/sec

A largura de banda é aproximadamente 968 vezes maior do que a seção anterior. Este aumento ocorre graças ao fato de que agora o nosso *switch* aprende com as regras do controlador, e recorda-se destas por meio da tabela de *flows*. Assim, após os pacotes iniciais, os demais pacotes não precisam passar pelo controlador, sendo encaminhados diretamente pelo *switch*, aumentando bastante a largura de banda. O funcionamento de nosso código pode ser percebido pelo uso do mesmo experimento feito nas seções

anteriores, porém desta vez enviaremos dois `pings`. O *output* do `tcpdump` dos computadores virtuais *h2* e *h3* são exibidos a seguir:

h2, primeiro ping:

```
11:13:10.697158 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
11:13:10.697201 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
11:13:11.688110 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
11:13:11.688152 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0001 0a00 0001                 ..........
```

h3, primeiro ping:

```
11:13:10.697155 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
11:13:11.688106 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
```

h2, segundo ping:

```
11:15:31.005714 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 14284, seq 1, le
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 80e9 4000 4001 a5bd 0a00 0001 0a00  .T..@.@.........
        0x0020:  0002 0800 84c5 37cc 0001 5327 bf5d 0000  ......7...S'.]..
        0x0030:  0000 6a15 0000 0000 0000 1011 1213 1415  ..j.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
11:15:31.005747 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 14284, seq 1, leng
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 884a 0000 4001 de5c 0a00 0002 0a00  .T.J..@..\......
        0x0020:  0001 0000 8cc5 37cc 0001 5327 bf5d 0000  ......7...S'.]..
```

```
        0x0030:  0000 6a15 0000 0000 0000 1011 1213 1415  ..j.............
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ..........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

h3, segundo ping:

((Output vazio))

Podemos perceber que, no primeiro `ping`, o computador virtual *h1* sequer obteve resposta. Isto ocorreu devido ao fato de termos comentado a função `resend_packet` no código do controlador. O primeiro serve apenas para adicionarmos os dados dos pacotes envolvendo *h1* e *h2* na tabela de *flows*. No segundo `ping`, temos que o computador *h1* obtém resposta, e é observável tráfego apenas no computador *h2*.

Podemos observar a existência dos *flows* de *h1* para *h2* e vice-versa, também, utilizando-se do comando `sudo ovs-ofctl dump-flows s1`:

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=15.352s, table=0, n_packets=2, n_bytes=140, idle_age=0,
 cookie=0x0, duration=16.337s, table=0, n_packets=3, n_bytes=182, idle_age=0,
```

8 Passo 5

Explique a lógica implementada no seu controlador `firewall.py` e mostre saídas de comandos que comprovem que ele está de fato funcionando (saídas dos comandos `tcpdump`, `sudo ovs-ofctl`, `nc`, `iperf` e `telnet` são recomendadas)

As regras especificadas no arquivo `fw.json` são convertidas para uma lista de dicionários, estes contendo as configurações pertinentes de cada uma das regras.
Antes mesmo de comparar qualquer regra, criamos um `ofp_match`. Começamos especificando o tipo do pacote (por meio do campo `dl_type`), pois especificar este campo de camada 2 é necessário para permitir a especificação de outros campos pertinentes de camada 3. Então conferimos o tipo do pacote para determinar as seguintes especificações: se o pacote for do tipo `ip`, podemos especificar o endereço ip da fonte (campo `nw_src`) e do destino (campo `nw_dst`); e se o pacote for do tipo `TCP` ou `UDP`, o seu protocolo é especificado (campo `nw_proto`), e é especificada também a porta de origem da camada de transporte (campo `tp_src`).

Após estas especificações serem feitas, é feita a comparação de cada um dos campos do pacote com os campos pertinentes de cada uma das regras. Se um pacote atender todas as especificações de uma ou mais regras, é acionada uma *flag* que faz com que futuros pacotes com estas especificações sejam descartados. Este comportamento é obtido simplesmente evitando de passar qualquer ação para o *flow* correspondente a estas especificações.

Para comprovar o funcionamento do firewall, faremos dois testes, com regras diferentes.
O primeiro teste consiste em modificar o arquivo `fw.json` para conter as seguintes especificações:

```
{
    "rule1" : {
        "ip_source": "10.0.0.1",
        "ip_dest": "10.0.0.3"
    }
}
```

Estas especificações fazem com que todo tráfego cuja origem seja o computador virtual *h1*, e o destino seja o computador virtual *h3*, seja bloqueado. Isto pode ser observado utilizando-se do comapndo `pingall`. Após o primeiro destes comandos (feito para adicionar os *flows* no *switch*), observamos o seguinte resultado:

```
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> X h2
*** Results: 33% dropped (4/6 received)
```

Adicionalmente, podemos observar a tabela de *flows*:

```
 cookie=0x0, duration=119.486s, table=0, n_packets=3, n_bytes=126, idle_age=68
 cookie=0x0, duration=144.5s, table=0, n_packets=5, n_bytes=210, idle_age=68,
 cookie=0x0, duration=99.457s, table=0, n_packets=3, n_bytes=126, idle_age=68,
 cookie=0x0, duration=89.443s, table=0, n_packets=3, n_bytes=126, idle_age=58,
 cookie=0x0, duration=114.473s, table=0, n_packets=5, n_bytes=210, idle_age=58
 cookie=0x0, duration=134.498s, table=0, n_packets=5, n_bytes=210, idle_age=68
 cookie=0x0, duration=143.46s, table=0, n_packets=2, n_bytes=196, idle_age=73,
 cookie=0x0, duration=104.468s, table=0, n_packets=1, n_bytes=98, idle_age=73,
 cookie=0x0, duration=133.504s, table=0, n_packets=2, n_bytes=196, idle_age=73
 cookie=0x0, duration=94.449s, table=0, n_packets=2, n_bytes=196, idle_age=63,
 cookie=0x0, duration=113.474s, table=0, n_packets=2, n_bytes=196, idle_age=63
 cookie=0x0, duration=124.446s, table=0, n_packets=2, n_bytes=196, idle_age=73
```

Dando destaque à nona entrada da tabela, percebemos que suas especificações de endereço IP de entrada e saída são correspondentes aos especificados na configuração, e a ação atríbuida a estas especificações é descartar o pacote. Como um complemento deste primeiro pacote, ao rodar o comando `iperf`, é feita uma tentativa de medir a largura de banda entre *h1* e *h3*. No entanto, o teste nunca é concluído, pois o envio de pacotes de *h1* para *h3* é impossibilitado.

O segundo teste que será feito consiste em modificar o arquivo `fw.json` para conter as seguintes especificações:

```
{
    "rule1" : {
```

```
        "protocol": "TCP"
    }
}
```

Estas especificações fazem com que todo tráfego cujos pacotes possuam em si o protocolo `TCP` seja descartado. O primeiro teste que faremos será utilizando-se do anteriormente mencionado `iperf`. Novamente, notamos que o programa encontra-se impossibilitado de terminar. Analisemos a tabela de *flows*:

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=70.301s, table=0, n_packets=1, n_bytes=42, idle_age=69,
 cookie=0x0, duration=69.304s, table=0, n_packets=2, n_bytes=148, idle_age=63,
```

Notamos a única especificação envolvendo pacotes `TCP` presente possui consigo a instrução de descartar estes pacotes. Vale notar que, por mais que hajam especificações adicionais neste *flow*, todo e qualquer pacote com especificações diferentes que carregasse consigo o protocolo `TCP` teria um novo *flow* adicionado para suas especificações, com as mesmas instruções de descarte. Para ilustrar este fato, abrimos terminais para os computadores virtuais *h2* e *h3*, utilizando neste último o comando `sudo lsof -i -P -n | grep LISTEN`, e obtivemos o seguinte resultado:

```
socat   2098  root  15u IPv4    18171       0t0 TCP *:6010 (LISTEN)
```

No computador *h2*, então, nos utilizamos do `telnet` para tentar nos conectar com o computador *h3*, o que resultou, novamente, em falha. Observando, novamente, a tabela de *flows*, temos o seguinte resultado:

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=136.419s, table=0, n_packets=1, n_bytes=42, idle_age=135
 cookie=0x0, duration=18.453s, table=0, n_packets=1, n_bytes=42, idle_age=17,
 cookie=0x0, duration=17.466s, table=0, n_packets=2, n_bytes=148, idle_age=11,
 cookie=0x0, duration=135.427s, table=0, n_packets=2, n_bytes=148, idle_age=12
```

Assim, é evidente o comportamento descrito anteriormente.

9 Configuração dos computadores virtual e real usados nas medições (se foi usado mais de um, especifique qual passo foi feito com cada um)

Todas as medições foram realizadas utilizando-se de um computador com as seguintes especificações:

- Modelo: Acer Aspire M5 series Z09
- Sistema Operacional: Ubuntu 18.04.3 LTS
- Memória RAM: 4GB
- Processador: Intel® Core™ i3-3227U CPU @ 1.90GHz 4

10 Referências

- https://wiki.wireshark.org/OpenFlow
- https://unix.stackexchange.com/questions/184965/open-file-from-remote-com
- https://openflow.stanford.edu/display/ONL/POX+Wiki.html#POXWiki-MatchStru
- http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt